Task-Oriented ML/DL Library Recommendation based on a Knowledge Graph

Mingwei Liu, Chengyuan Zhao, Xin Peng, Simin Yu, Haofen Wang, and Chaofeng Sha

Abstract—AI applications often use ML/DL (Machine Learning/Deep Learning) models to implement specific AI tasks. As application developers usually are not AI experts, they often choose to integrate existing implementations of ML/DL models as libraries for their AI tasks. As an active research area, AI attracts many researchers and produces a lot of papers every year. Many of the papers propose ML/DL models for specific tasks and provide their implementations. However, it is not easy for developers to find ML/DL libraries that are suitable for their tasks. The challenges lie in not only the fast development of AI application domains and techniques, but also the lack of detailed information of the libraries such as environmental dependencies and supporting resources. In this paper, we conduct an empirical study on ML/DL library seeking questions on Stack Overflow to understand the developers' requirements for ML/DL libraries. Based on the findings of the study, we propose a task-oriented ML/DL library recommendation approach, called MLTaskKG. It constructs a knowledge graph that captures AI tasks, ML/DL models, model implementations, repositories, and their relationships by extracting knowledge from different sources such as ML/DL resource websites, papers, ML/DL frameworks, and repositories. Based on the knowledge graph, MLTaskKG recommends ML/DL libraries for developers by matching their requirements on tasks, model characteristics, and implementation information. Our evaluation shows that 92.8% of the tuples sampled from the resulting knowledge graph are correct, demonstrating the high quality of the knowledge graph. A further experiment shows that MLTaskKG can help developers find suitable ML/DL libraries using 47.6% shorter time and with 68.4% higher satisfaction.

Index Terms—Machine Learning, Deep Learning, Library Recommendation, Knowledge Graph.

1 Introduction

NOWADAYS, more and more software applications integrate some kinds of AI capabilities to implement innovative requirements, *e.g.,* intelligent human-computer interaction [1], [2], [3], recommendation [4], [5], and decision making [6], [7]. These AI applications often use ML/DL (Machine Learning/Deep Learning) models to implement specific AI tasks such as text classification [8], [9], facial recognition [10], [11], and image classification [12], [13]. As application developers usually are not AI experts, they often choose to reuse existing implementations of ML/DL models. To this end, the developers need to find suitable ML/DL models for their AI tasks and integrate the implementations as libraries.

As an active research area, AI attracts many researchers and produces a lot of papers every year. For example, in 2019 more than 120 thousand peer-reviewed AI papers have been published in conferences and journals and the number keeps growing [14]. These papers cover major AI technical fields such as computer vision (CV) [15], [16], natural language processing (NLP) [17], [18], speech processing (SP) [19], [20], and autonomous driving (AD) [21], [22]. A large part of these papers target specific AI tasks, propose ML/DL models for these tasks, and provide their implementations as open-source projects. Application developers could reuse those open-source projects with implementations of ML/DL models in different ways. If the open-source project has a release package (*e.g.,* the PyPi package), it can be directly integrated and used by developers through APIs. If the open-source project does not provide a release package but provides the model implementation and the trained model, developers can integrate the model implementation into their projects and directly reuse the trained model. In other cases, developers need to retrain the model on top of the integrated model implementation. In this paper, we define ML/DL libraries as projects that provide ML/DL model implementations for AI tasks.

Although many ML/DL models and implementations are available, it is not easy for application developers to find and use ML/DL libraries that are suitable for their AI tasks. The challenges lie in two aspects. First, AI application domains and techniques are still in rapid development and new innovative tasks keep emerging. For example, sentence simplification [23], [24], which simplifies the grammar and structure of a sentence while keeping the underlying information identical, is a relatively new text processing task as a variant of text summarization. Unlike popular tasks (*e.g.,* text classification) which have well-known ML/DL models and libraries (*e.g.,* fastText [25]), models and libraries for these new tasks may only be presented in a few papers and are not easy to find. Second, developers often need to consider a series of factors when choosing ML/DL libraries. For example, they may be concerned about the performance on recognized indicators, important components (*e.g.,* CNN, RNN, and attention), environmental dependencies (*e.g.,* language, ML/DL frameworks, OS, and hardware), and the availability of datasets, trained models, and test code. This detailed information of the libraries is scattered and implied in different sources (*e.g.,* papers, documentation, and source code), thus is hard to collect.

In the AI community researchers have built through

- *M. Liu, C. Zhao, X. Peng, S. Yu, C. Sha are with the School of Computer Science and Shanghai Key Laboratory of Data Science, Fudan University, China.*
- *X. Peng is the corresponding author (pengxin@fudan.edu.cn).*
- *H. Wang is with Tongji University, China (carter.whfcarter@gmail.com).*

crowdsourcing PapersWithCode[1], a free and open resource with ML/DL papers and code, but it is hard for application developers to find the desired models and libraries from PapersWithCode. First, its AI task categories cannot well match the targeted tasks of application developers. PapersWithCode provides a hierarchy of AI tasks, but some of the tasks are not specific enough for application development. For example, software named entity recognition is not defined as a task and related papers are categorized into the more general task named entity recognition. Second, it lacks detailed information about the implementations of ML/DL models. PapersWithCode provides links to open-source repositories for some SOTA (State-Of-The-Art) models, but does not collect important information required by application developers such as environmental dependencies. There are also open-source projects (called AwesomeLists) that provide curated lists of resources dedicated to specific AI technical fields (*e.g.*, awesome-nlp[2]). These lists include some recommended libraries for popular tasks, but only cover a small part of the tasks that application developers require.

To understand the requirements of application developers for ML/DL libraries, we conduct an empirical study on a set of sampled Stack Overflow questions about ML/DL models and implementations. The results show that the developers are concerned about a variety of factors when seeking solutions for their AI tasks. These factors are related to both the ML/DL models (*e.g.*, ML/DL components, model performance, and datasets) and their implementations (programming language, open source, and ML/DL frameworks).

Based on the findings of the study, we propose a task-oriented ML/DL library recommendation approach, called MLTaskKG. It constructs a knowledge graph (KG for short) that captures AI tasks, ML/DL models, model implementations, and their relationships by extracting knowledge from PapersWithCode, AwesomeLists, ML/DL papers, ML/DL frameworks, and open-source repositories. Based on the KG, MLTaskKG recommends ML/DL libraries for application developers by matching their requirements on tasks, model characteristics, and environmental dependencies with the knowledge collected in the KG. Note that while MLTaskKG may be especially useful for application developers who are beginners in the ML/DL domain, it can also be helpful for experienced practitioners who are seeking to implement models in new fields or tasks.

We implement MLTaskKG and build an AI task-model knowledge graph, which includes 17,250 AI tasks, 25,404 papers, 25,718 models, and 24,047 repositories. Our evaluation shows that the quality of the resulting knowledge graph is generally high, indicated by the correctness of 92.8% for the sampled tuples. Further, our tool can help developers find suitable ML/DL with 68.4% higher satisfaction and using 47.6% shorter time compared with those using PapersWithCode.

Significance. Our AI task-model KG links AI tasks, ML/DL models, and their implementations in a knowledge-based way and captures the environmental dependencies and supporting resources of the implementations. It thus can help to bridge the gap between AI researchers and application developers and facilitate the large-scale reuse of ML/DL models and libraries.

2 Background

A typical form of AI papers is to design and implement a new ML/DL model for a specific AI task. The task can be a well-studied one, which has much previous research or an innovative one for a new application domain or scenario. The model is often designed based on some standard components such as neural networks (*e.g.*, CNN, RNN) and additional layers (*e.g.*, attention). The paper usually describes the targeted task and the proposed model (with a specific name) in its title and abstract. To validate the proposed model, the authors usually implement the proposed model and compare it with existing ones. To facilitate comparison and reproduction, the authors often share their implementation in repositories hosted on open-source platforms such as GitHub and provide a link to the repository in the paper.

The authors of a paper often implement their model based on a common ML/DL framework such as TensorFlow[3] and PyTorch[4] and the implementation may depend on a specific version of the framework. Moreover, the implementation may depend on some third-party libraries. A version of an ML/DL framework may have specific requirements on hardware and operating system. For example, PyTorch does not support Windows versions lower than 7. To help users deploy and run their implementations, the authors may describe various usage guidelines in the ReadMe file of the repository, *e.g.*, the dependencies of the model implementation and guidelines for training and prediction. If a trained model is available, the ReadMe file may also provide a link for download. In addition to the implementation provided by the authors, other people may also reproduce the model and share their implementations in open-source repositories. Some standard implementations of popular tasks such as text classification and image classification may also be included in some common ML/DL libraries (*e.g.*, fastText).

AwesomeLists are a special kind of projects on GitHub, which provide curated lists of resources dedicated to different topics. There are AwesomeLists for specific AI technical fields such as CV and NLP. For example, awesome-nlp lists books, courses, libraries, and other resources for NLP. For each of the recommended libraries, a note is provided describing the implemented tasks and the implementation language.

PapersWithCode is a website for sharing ML/DL papers and code. It defines a hierarchy of AI tasks by crowdsourcing and the top-layer tasks include CV, NLP, SP, and others. The main users of PapersWithCode are AI researchers and one of its main purposes is to help researchers learn related papers and compare with the SOTA models. Therefore, for each task it collects a set of datasets and for each dataset provides an evaluation table providing a set of SOTA models ranked by specific metrics together with the papers. For each paper with code, it provides links to the corresponding repositories.

3 Empirical Study

We conducted an empirical study on a set of sampled Stack Overflow questions about ML/DL models and implementa-

1. https://paperswithcode.com
2. https://github.com/keon/awesome-nlp
3. https://www.tensorflow.org
4. https://pytorch.org

TABLE 1
Factors for Selecting ML/DL Libraries

Factor	Definition	Subject	Example	#Q
Programming Language	the programming languages used by the implementation	Implementation	Python, Java	151
Open Source	whether the implementation is open-sourced	Implementation	open-source implementation	49
Model Performance	the performance of the model on the target task	Model	best model	45
ML/DL Framework	the ML/DL framework that the implementation is based on	Implementation	PyTorch, TensorFlow	43
Operating System	the operating systems that the implementation supports	Implementation	Android, iOS	37
ML/DL Component	common ML/DL components that are included in the model	Model	CNN, LSTM, attention	32
Hardware	the hardware that the implementation supports	Implementation	GPU	28
Usability	the usability of the implementation	Implementation	easy to use	25
Model Characteristic	the characteristic of the model	Model	real-time prediction	21
Dataset	the publicly available datasets that can be used for the model	Model	ImageNet	10
Implementation Quality	the quality of the implementation	Implementation	well-maintained	9

tions to understand the requirements of application developers. This section reports the data preparation, protocol, and results of the study.

3.1 Data Preparation

The focus of the empirical study is on the developers' requirements for ML/DL libraries. Therefore, we use the following criteria to select Stack Overflow questions for the study:

1) tagged with "machine-learning", "deep-learning", or any of the known AI tasks;

2) question title or body containing any of the known AI tasks;

3) question body containing no code snippets (*i.e.,* text wrapped in `<pre><code></code></pre>`);

4) question title and body containing no "error" or "exception".

We first obtain questions that meet all the above criteria from the Stack Overflow data dump [26] as a population, resulting in 68,501 questions. We then randomly select 1,000 questions from the population for the study. The first two criteria are to ensure that the sampled questions are related to AI tasks. The known AI tasks here mean the tasks that are collected in the task categories of PapersWithCode, which are included in our replication package [27]. The last two criteria are to filter out questions about more detailed implementation issues.

Then we invite two master students (not authors) to independently check the selected questions to filter out those that are not related to the seeking of ML/DL libraries. The two students both have taken at least two courses on AI and have at least one year of AI development experience. The manual checking results in 283 questions that are confirmed to be relevant by both students with a Cohen's Kappa agreement [28] of 0.664 and squared Kappa of 0.441, *i.e.,* moderate agreement. We found that disagreements often occurred when the students overlooked some parts of very long questions or students had a different understanding of AI-related knowledge. One of the authors was assigned to resolve the conflict by a majority-winning strategy after the discussion.

We find most of these questions focus on major AI tasks such as computer vision (41.7%), natural language processing (27.9%), speech processing (13.4%), and data mining (12.0%). Among these questions, 54.4% have no accepted answers. Although the final purposes of these questions are mostly some kind of implementations that can be used directly or after adaptation, some askers may first ask for ML/DL models (also mentioned as algorithms, approaches, and methods) for specific tasks and then learn about the implementations that can be used in the follow-up discussions. The others directly ask about the implementations (also mentioned as libraries, projects, and repositories) that can be used for specific AI tasks.

3.2 Protocol

The requirements of application developers for ML/DL libraries are embodied by a set of factors that are considered in the selection of ML/DL libraries. Therefore, we ask two master students (the same as students in Section 3.1 for question selection) to further annotate the factors for ML/DL library selection from the 283 Stack Overflow questions using iterative closed coding. Note that the requirements of a developer are reflected not only in the question title and body but also in the follow-up discussions. For example, the asker of a question may comment on a suggested solution to express additional concerns. Therefore, we ask the two annotators to read both the question title/body and the follow-up discussions to have a complete understanding of the requirements of the asker.

The coding starts with an initial code, *i.e.,* model performance, which means the performance of the model on the target task. For a Stack Overflow question the annotators may identify multiple factors. If an identified factor is not included in existing codes, we create a new code or revise the definitions and names of existing codes to accommodate it. When a new code is created or an existing code is revised, we re-annotate all the questions that have been annotated before. The two annotators conduct the annotation independently. If they identify different factors for the same question, a third student is assigned to make an additional annotation to resolve the conflict by a majority-win strategy.

3.3 Results

The study results in 11 factors for ML/DL library selection with a Cohen's Kappa coefficient 0.810 and squared Kappa of 0.656, *i.e.,* almost perfect agreement. We found the disagreements often occurred when the students overlooked some

parts of very long questions, which can easily be resolved through discussion. Table 1 shows the information of the 11 factors, including definition, subject, example, and the number of questions involving the factor.

It can be seen that the developers are concerned about a variety of factors when seeking solutions for their AI tasks. Four factors are related to the ML/DL models. Among them, model performance (15.9%), ML/DL component (11.3%) and model characteristic (7.4%) are the three most concerned factors. The other seven factors are related to the implementations of the models. Among them, programming language (53.4%), open source (17.3%), and ML/DL framework (15.2%) are the three most-cited factors.

From the questions, we can see that developers have different concerns and preferences. Some developers care about the model performance (*e.g.,* SOTA models) and used common components (*e.g.,* LSTM). Some others do not care about these, but those factors related to the deployment and usage of the implementations such as programming language, AI framework, and other dependencies. Therefore, it is necessary to incorporate knowledge of various levels (*e.g.,* model, implementation) and support application developers with task-oriented and knowledge-based ML/DL library recommendation.

4 AI Task-Model KG Construction

MLTaskKG is based on a semi-automatically constructed AI task-model KG. In this section, we describe the approach for the KG construction and then introduce the resulting KG.

4.1 Overview

The AI task-model KG is designed for ML/DL library selection and recommendation, so the KG needs to describe the relationships between AI tasks, ML/DL models, and their implementations and cover those factors identified in the empirical study. Therefore, we design the schema of the KG as shown in Figure 1. In the figure, rounded rectangles represent different types of entities in the KG and arrows represent the relationships between different types of entities.

AI tasks constitute a conceptual hierarchy based on their generalization/specialization relationships. For example, extractive text summarization is a specialization of text summarization. An AI task can be accomplished by an ML/DL model which is proposed in a paper and may have been evaluated with specific datasets. An ML/DL model may use existing ML/DL components (*e.g.,* CNN, LSTM, attention) and has its implementation. The implementation is often based on a specific version of a common ML/DL framework (*e.g.,* PyTorch, TensorFlow), and supports specific programming languages and running environments such as hardware (*e.g.,* GPU) and operating systems (*e.g.,* Android, iOS). The implementation may be provided in an open-source repository with a specific license. Besides relationships, each entity type can have some attributes. For example, a repository has attributes such as stars and URLs.

To construct such a KG, we design an approach that extracts various entities and relationships from different sources and fuses them together. Figure 2 presents an overview of the approach. The top layer, middle layer, and bottom layer of the figure describe the knowledge sources, the main steps, and the corresponding parts of the KG, respectively. The approach uses the following knowledge sources.

- **PapersWithCode**: A website for sharing ML/DL papers and code, which includes a set of AI tasks and the corresponding papers and datasets.
- **ML/DL papers**: Papers that propose ML/DL models for specific AI tasks.
- **Repositories**: Open-source repositories that provide the implementations for ML/DL models.
- **AwesomeLists**: A special kind of projects on GitHub, which include curated lists of libraries for specific AI tasks.
- **ML/DL Framework Docs**: The documentations of ML/DL frameworks, which describe the required running environments for the frameworks.

The approach includes three steps. **Task-model Knowledge Extraction** extracts knowledge about tasks and models from both PapersWithCode and ML/DL papers. PapersWithCode has already defined a hierarchy of AI tasks with a large number of AI tasks. For each task, it collects one or several SOTA models for different datasets together with the corresponding papers. We parse the PapersWithCode data and extract this knowledge. Based on the knowledge, we further analyze a large corpus of ML/DL papers to extract more AI tasks and models. We combine these two parts of AI tasks to form the AI task hierarchy of the KG. **Framework Knowledge Extraction** extracts knowledge about the running environments required by different versions of popular ML/DL frameworks (*e.g.,* PyTorch, TensorFlow). This step is manually done based on the documentation of the frameworks. Based on the extracted task-model knowledge and framework knowledge, **Repository Knowledge Extraction** extracts knowledge about the implementations of ML/DL models and the corresponding open-source repositories. The data analyzed include the metadata (*e.g.,* repository star, last update time), ReadMe files, and source code of the repositories.

The knowledge (including entities and their relationships) extracted from different sources is linked together following the schema shown in Figure 1, resulting in the AI task-model KG.

4.2 Task-Model Knowledge Extraction

We first extract basic knowledge about AI tasks and ML/DL models from PapersWithCode, then extend the knowledge by analyzing ML/DL papers, and finally integrate these two parts of knowledge.

4.2.1 Task-Model Knowledge Extraction from PapersWithCode

The knowledge extraction from PapersWithCode includes two parts. First, we develop a web crawler using Scrapy[5] to extract the AI tasks and their hierarchical relationships from the website of PapersWithCode. Second, we analyze the daily updated data dump of PapersWithCode[6] to extract

5. https://github.com/scrapy/scrapy
6. https://github.com/paperswithcode/paperswithcode-data

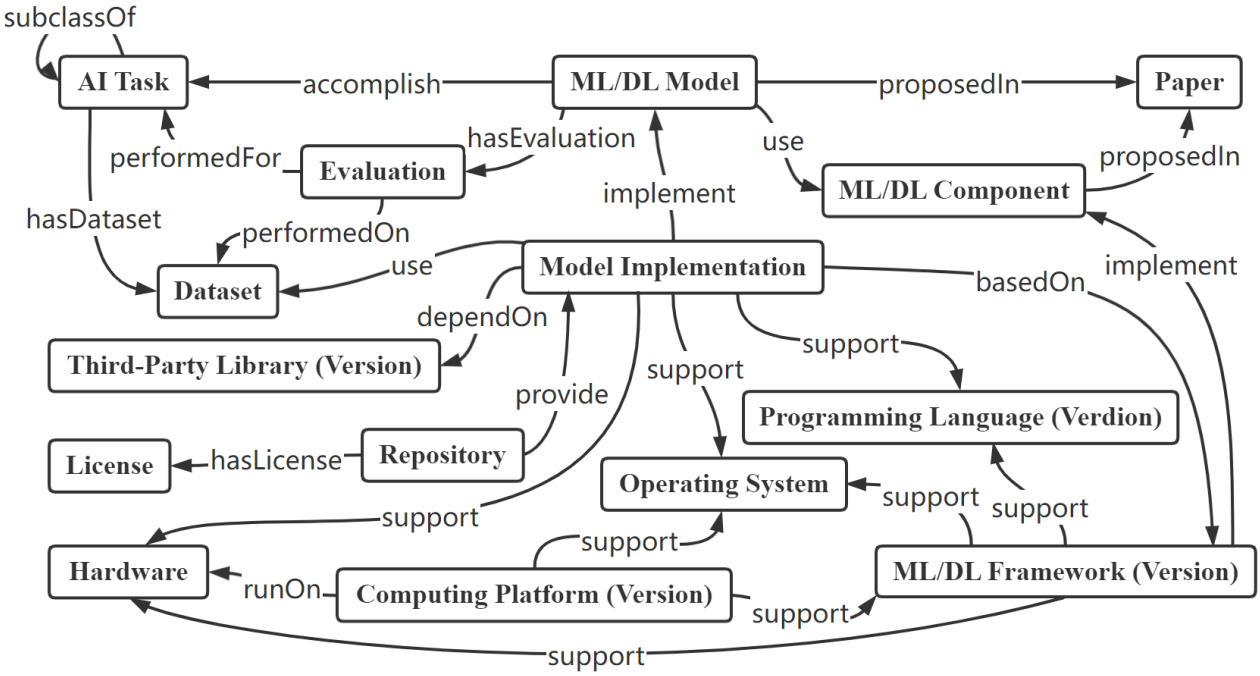


Fig. 1. AI Task-Model KG Schema

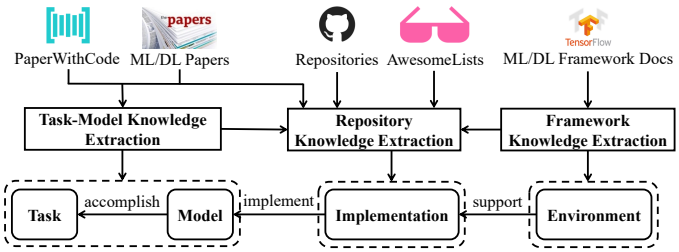


Fig. 2. Overview of AI Task-Model KG Construction

related datasets and SOTA models, and common ML/DL components. The data dump provides a list of datasets for each AI task and for each dataset an evaluation table with a set of SOTA models ranked by specific metrics. For example, MatchSum is a SOTA model for the text summarization task, which is ranked first for the BBC XSum dataset by the metric ROUGE. The data dump also includes a set of common ML/DL components such as CNN, RNN, LSTM, and BiLSTM. We organize all the above knowledge based on the schema shown in Figure 1.

4.2.2 *Task-Model Knowledge Extraction from ML/DL Papers*

The purpose of ML/DL paper analysis is to extend the tasks and models extracted from PapersWithCode. We crawl a corpus of ML/DL papers from major AI conferences, including CVPR, ICCV, ECCV, ACL, and EMNLP, for further analysis. These papers were published from 2011 to 2020. A reason for choosing these conferences is that their papers focus more on specific AI tasks of different domains (*e.g.*, CV, NLP). For each paper, we extract its title, abstract, authors, conference, year of publication, and citation number. If the full text (PDF format) of a paper is available, we also download the full text. As a result, we obtain 20,653 papers and 14,200 of them have full text.

ML/DL papers often highlight the targeted AI tasks and proposed ML/DL models in their titles or abstracts. For example, the title "AttentionXML: Label Tree-based Attention-Aware Deep Model for High-Performance Extreme Multi-Label Text Classification" of a paper includes the targeted task "multi-label text classification" and the name of the proposed model "AttentionXML". Therefore, we use a deep learning model to extract AI tasks and ML/DL models from the titles and abstracts.

We treat the problem as a named entity recognition (NER) task, where AI tasks and ML/DL models are the two types of entities to be recognized. A common solution for the task is using a sequence tagging model which accepts a word sequence as input and predicts a tag for each word. Named entities can then be recognized by interpreting the tags of words. We use the BIO tagging scheme: "B" ("Beginning") and "I" ("Inside") respectively indicate the beginning and middle/end of an entity, and "O" ("Outside") indicates a normal word. Thus, our NER task defines the following five tags: "B-Task" and "I-Task" indicate the begging and middle/end of a subsequence for an AI task; "B-Model" and "I-Model" indicate the begging and middle/end of a subsequence for an ML/DL model; "O" indicates a normal word that is not a part of the task or model. For the title mentioned above, the tagging of the whole word sequence is shown in Figure 3. "AttentionXML" is tagged with "B-Model" (*i.e.*, the blue part), while "multi-label text classification" is tagged with "B-Task I-Task" (*i.e.*, the green part).

We implement and train a sequence tagging model BERT-BiLSTM for the task. It includes three layers: BERT (Bidirectional Encoder Representations from Transformers) layer, Bi-LSTM (Bidirectional Encoder Representations from

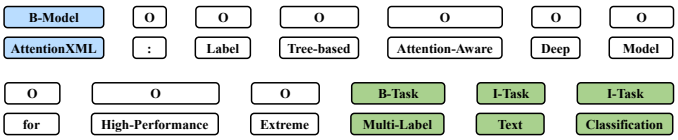


Fig. 3. An Example of NER Tagging

Transformers) layer, and FC (Fully Connected) layer. The BERT layer is an embedding layer that converts each input word to a fixed-size vector using the pretrained language model [29]. BERT is pretrained on a large corpus and its vector representation can well capture the semantics of a word in different contexts. The Bi-LSTM layer further encodes and combines the vectors of a word and its left context and right context to prepare the input for the FC layer. The FC layer performs the final prediction based on the encoded vectors of all words. We implement the model based on PyTorch 1.8.1 and use SciBERT [30] for the BERT layer, which is pretrained on scientific text from papers and achieves the SOTA performance on a wide range of NLP tasks in the scientific domain. The SciBERT model used in our implementation is obtained from HuggingFace[7].

Given an ML/DL paper, we concatenate its title and abstract as the input word sequence. To indicate the two different parts in the sequence, we add a label "[TITLE]" and "[ABSTRACT]" at the beginning of the title and abstract respectively. To train the sequence tagging model we ask four master students to annotate a set of sampled ML/DL papers by identifying the AI tasks and ML/DL models mentioned in the titles and abstracts. The four students have taken at least two courses on AI and have at least one year of AI development experience. The annotation is done with an online annotation tool doccano [31]. As a result, we obtain 800 annotated papers for the training.

We use the trained model to predict the tags of words of the titles and abstracts of all the other papers and extract the subsequences that are tagged as tasks or models. We treat the extracted tasks and models as the recognized AI tasks and ML/DL models of the given papers. We conducted experiments on a test dataset to evaluate the accuracy of our model in extracting AI tasks and ML/DL models. The accuracy of task extraction was found to be 0.74 and the accuracy of model extraction was 0.82. These results indicate an acceptable performance, considering the complexity and variability of natural language texts.

4.2.3 Task-Model Knowledge Integration

The basic task structure extracted from PapersWithCode is treated as the task hierarchy of the KG. Then we take an iterative process to incorporate each of the AI tasks extracted from the papers into the task hierarchy. For each AI task extracted from papers, we first match it with the existing task hierarchy. If the task matches an existing task (full name or aliases), we merge them together. Otherwise, we create a new AI task and insert it into the task hierarchy. As the hierarchy of AI tasks defined by PapersWithCode has covered most of the popular tasks, it is likely that the new task extracted from papers is the specialization of an existing task. Therefore, we decide the position of the new task in the task hierarchy in the following way. For a new task T, we calculate its similarity with each of the existing tasks in the task hierarchy. The similarity is obtained by calculating the cosine similarity between the vector representations of two tasks generated by averaging their word vectors. The word vectors are produced by a 100-dimensional Word2Vec model trained on the abstracts of all the papers we collected. We train the Word2Vec model by using gensim [32], whose dimensions are determined based on conventions in NLP[8]. Based on the similarity calculation, we find an existing task T' having the highest similarity with T as the candidate for the following processing.

1) If the similarity between T and T' is lower than a threshold (*e.g.*, 0.75 in our implementation), T is not similar enough to a specific task of the existing task hierarchy and thus can only be added as a specialization of one of the top-layer tasks (*e.g.*, CV, NLP) according to the domain of the corresponding paper (*e.g.*, ICCV papers belong to the CV domain).

2) If the similarity is no lower than the threshold and T' is a substring of T, T can be added as a specialization of T'. For example, an AI task "Software Named Entity Recognition" extracted from papers is added as the specialization of "Named Entity Recognition" in the task hierarchy.

3) If the similarity is no lower than the threshold and T' is not a substring of T, T can be treated as a sibling task of T' and added as a specialization of the parent task of T'.

4.3 Framework Knowledge Extraction

We collect 18 popular ML/DL frameworks, including PyTorch, TensorFlow, DeepLearning4J, Keras, Caffe, CNTK, MXNet, Theano, TFLearn, PaddlePaddle, Chainer, Neon, Lasagne, Sonnet, BigDL, Dlib, PlaidML, and SINGA. These frameworks cover different programming languages, *e.g.*, TensorFlow and PyTorch are for Python and DeepLearning4j is for Java. For each of them, we manually analyze its documentation to identify its versions, *e.g.*, PyTorch 1.4.0. For different versions of the framework, we further extract the following knowledge.

- **Programming Language**: programming languages in which the framework is developed, *e.g.*, Python, Java.
- **Operating System**: systems on which the framework can be installed, *e.g.*, CentOS, Debian, macOS, Android, iOS.
- **Hardware**: physical components supported by the framework, *e.g.*, RTX 3060, GTX 1070.
- **Computing Platform**: a platform for parallel computing of the framework, *e.g.*, CUDA, ROCm.
- **ML/DL components**: common ML/DL components that have been implemented in the framework.

Note that environmental dependencies are usually related to specific versions. For example, CUDA 10.1 supports PyTorch 1.4.0; PyTorch 1.3.0 and higher versions support Android and iOS.

7. https://huggingface.co/allenai/scibert_scivocab_uncased

TABLE 2
Implementation Knowledge Types and Their Sources

Knowledge Type	Knowledge Source	Corresponding Factor	Knowledge Type	Knowledge Source	Corresponding Factor
Star Number	Meta	Implementation Quality	Fork Number	Meta	Implementation Quality
Watch Number	Meta	Implementation Quality	Issue Number	Meta	Implementation Quality
License	Meta	Open Source	Homepage URL	Meta	Implementation Quality/Usability
Last Update Time	Meta	Implementation Quality	Operating System	ReadMe	Operating System
Hardware	ReadMe	Hardware	Dataset	ReadMe	Dataset
ML/DL Framework	Code, ReadMe	ML/DL Framework	Programming Language	Meta, ReadMe	Programming Language
Third-Party Library	Code, ReadMe	Implementation Quality, Third-Party Library	Release Package	ReadMe	Implementation Quality/Usability
Trained Model	ReadMe	Usability	Command	ReadMe	Usability
ML/DL Component	Code	ML/DL Component	Documents	Code	Usability
Test Code	Code	Implementation Quality	Example Script	Code	Usability

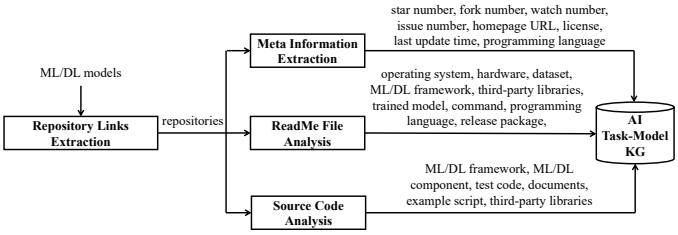


Fig. 4. Overview of Repository Knowledge Extraction

4.4 Repository Knowledge Extraction

To extract knowledge about implementations, we first obtain the repository links of ML/DL model implementations and then extract various knowledge from different sources. An overview of the repository knowledge extraction process is shown in Figure 4. Table 2 shows the sources of different types of implementation knowledge and the corresponding factors for selecting ML/DL libraries (see Table 1). The implementation knowledge is extracted from the following three sources.

- **Meta**: the meta information of a repository, *e.g.*, start number, fork number.
- **ReadMe**: the ReadMe file of a repository.
- **Code**: the source code of a repository.

Moreover, the environmental dependencies of a model implementation can be inferred from those of an ML/DL framework. For example, if a model implementation is based on PyTorch 1.4.0, we can infer that the implementation supports Linux because PyTorch 1.4.0 supports Linux.

4.4.1 Repository Links Extraction

For each ML/DL model, we obtain the links of its repository from the corresponding paper using the following four strategies. Given a paper, we try to obtain the repository links for its model implementation in the following ways.

- **PapersWithCode**. If the paper is included in the data dump of PapersWithCode, the repository links may also be included. If the links are available, we obtain them directly.
- **Paper Abstract Analysis**. Some ML/DL papers mention the repository links in their abstracts. Therefore, we use regular expressions to find possible repository links mentioned in the abstract of the paper.
- **Paper Body Analysis**. Some ML/DL papers provide the repository links in the body (including footnotes) of the paper. We analyze the PDF file of the paper (if available) using PDFMiner[9] and treat the first URL to GitHub as the repository link.
- **GitHub Searching**. Some repositories implementing ML/DL models mention the titles of the corresponding papers in their descriptions. We search GitHub with the title of the paper using GitHub APIs to obtain the top-10 related repositories. Then we analyze the returned repositories and treat those mentioning the paper title in their ReadMe files as the repositories.

We develop a tool to obtain the repository link of a paper using the above four strategies. It tries the strategies in order and stops when a repository link is found. On the other hand, we extract 827 ML/DL libraries with GitHub repository links from the related AwesomeLists (*i.e.*, awesome nlp, awesome computer vision, awesome machine learning and awesome deep learning). These libraries provide recognized implementations for popular ML/DL models and AI tasks. As AwesomeLists does not explicitly state the ML/DL models that are implemented by each library, we need to recover the links between the library and the corresponding ML/DL models. We find the names of ML/DL models often appear in the class names of the corresponding implementations. Based on the findings, we identify ML/DL models that are implemented in a library by calculating the Jaccard similarity [33] between the class names of the library and the names of the ML/DL models in the KG. Before the similarity calculation, the preprocessing steps of the class name are as follows: camel case splitting, lowercase and lemmatization. If the similarity between the name of an ML/DL model and the class name with the highest Jaccard similarity is no lower than a threshold (*e.g.*, 0.4 in our implementations), we treat the library as an implementation of the model and add the relationship into the KG.

4.4.2 Meta Information Extraction

To extract the meta information of a repository, we use PyGitHub[10] (a Python library) to access the GitHub REST APIs. The extracted information includes star number, fork number, watch number, issue number, license, programming

8. https://moj-analytical-services.github.io/NLP-guidance/NNmodels.html

9. https://github.com/euske/pdfminer

10. https://github.com/PyGithub/PyGithub

language, homepage URL, and last update time. Our current implementation only supports GitHub, as it relies on GitHub APIs to get the meta information and ReadMe files of repositories. It is easy to extend the implementation to support other platforms if their APIs are available.

4.4.3 ReadMe File Analysis

GitHub repositories usually include a ReadMe file which contains important information about the deployment and usage of the software such as environmental dependencies [34]. Given a ReadMe file we extract implementation knowledge from it in the following two ways.

First, we extract implementation knowledge based on environment-related entities in the KG. These entities include the operating system, hardware, dataset, programming language, and ML/DL framework. If these entities are mentioned in the ReadMe file, we further use regular expressions to extract possible versions mentioned with the entities and add the corresponding relationships into the KG. For example, if "PyTorch" is mentioned we further extract the version "1.1.0" mentioned together with it and add a "support" relationship from the current model implementation to "PyTorch 1.1.0".

Second, we extract implementation knowledge using predefined patterns. ReadMe files often follow some patterns to describe certain implementation knowledge. These patterns involve not only linguistic patterns in the text but also the section structure, hyperlinks, and code blocks of the ReadMe file. For example, third-party library dependencies are often described after a title named "requirement", "dependency", "dependencies", or "environment". To collect such patterns we manually analyze 100 sampled ReadMe files and summarize a set of patterns for the extraction of the following knowledge.

- **Third-Party Library**: dependencies of third-party Libraries.
- **Release Package**: the release package of the current implementation, *e.g.,* a python package shared on PyPi[11].
- **Trained Model**: an instantiated ML/DL model trained using the implementation and certain dataset, which can be used directly.
- **Command**: commands that can be used to run the implementation, for example, for training a new model or load an existing model.

A complete list and descriptions of the patterns can be found in our replication package [27].

4.4.4 Source Code Analysis

From the source code of the implementation of an ML/DL model, we extract the following knowledge about the implementation.

- **ML/DL Framework**. We analyze the import statements of the source files. If an ML/DL framework is imported, we consider that the implementation is based on the framework.
- **Third-Party Library**. We analyze the dependencies declared in the implementation, *e.g.,* the "requirement.txt" of Python projects, to extract the dependencies on third-party libraries.
- **Documents**. We extract documents from the source directories named "doc" or "docs".
- **Example Script**. We extract example scripts from the source directories named "example" or "examples".
- **Test Code**. We extract test code from 1) directories named "test" or "tests"; source files whose names contain "test"; or 3) classes whose names contain "test".
- **ML/DL Component**. We first locate the major class that implements the model by calculating the Jaccard similarity [33] between the class name and model name. The class with the highest similarity is selected as the implementation class. We then analyze the used ML/DL components by identifying the classes that the implementation class depends on. If the names of the identified classes contain known ML/DL components (*e.g.,* CNN, LSTM) in the KG, we think the ML/DL components are used by the current model.

4.5 Resulting KG

The resulting AI task-model KG includes 159,310 entities and 628,045 relationships. The entities include 17,250 *tasks*, 25,404 *papers*, 25,718 *models*, 21,003 *model implementations*, and 24,047 *repositories*. Among them, 1,017 *tasks* and 9,839 *models* are obtained from PapersWithCode; 16,233 *tasks* and 15,805 *models* are extracted from ML/DL papers; 2,419 *model implementations* and 827 *repositories* are obtained from AwesomeLists. The relationships include 17,410 *subclassOf* relationships between tasks, 44,438 *accomplish* relationships, 20,594 *hasEvaluation* relationships, 29,281 *implement* relationships, 21,008 *provide* relationships, 60,040 *basedOn* relationships, and 105,963 *support* relationships. The complete data of the resulting KG is available in our replication package [27].

Figure 5 presents a part of the KG. It shows MatchSum, a BERT-based model proposed in ACL 2020[12] for extractive text summarization task. The KG also includes an implementation of the model. The implementation is based on PyTorch 1.4.0, depends on Python library transformer 2.5.1, and supports Python 3.7, GPU, and Linux.

5 KG-based Trend Analysis

For application developers and related researchers, it is difficult to have a comprehensive understanding of AI tasks, ML/DL models, and their implementations, because relevant information is scattered in different places. MLTaskKG links AI tasks, ML/DL models, and their implementations in a knowledge graph, thus can help to analyze the trends of AI tasks and their implementations. This analysis is the most direct application of our knowledge graph, and it can deepen our understanding of the problem scenarios addressed in this paper.

Design. With the continuous emergence of new applications, tasks, ML/DL models and model implementations, AI

11. https://pypi.org/

12. https://aclanthology.org/2020.acl-main.552

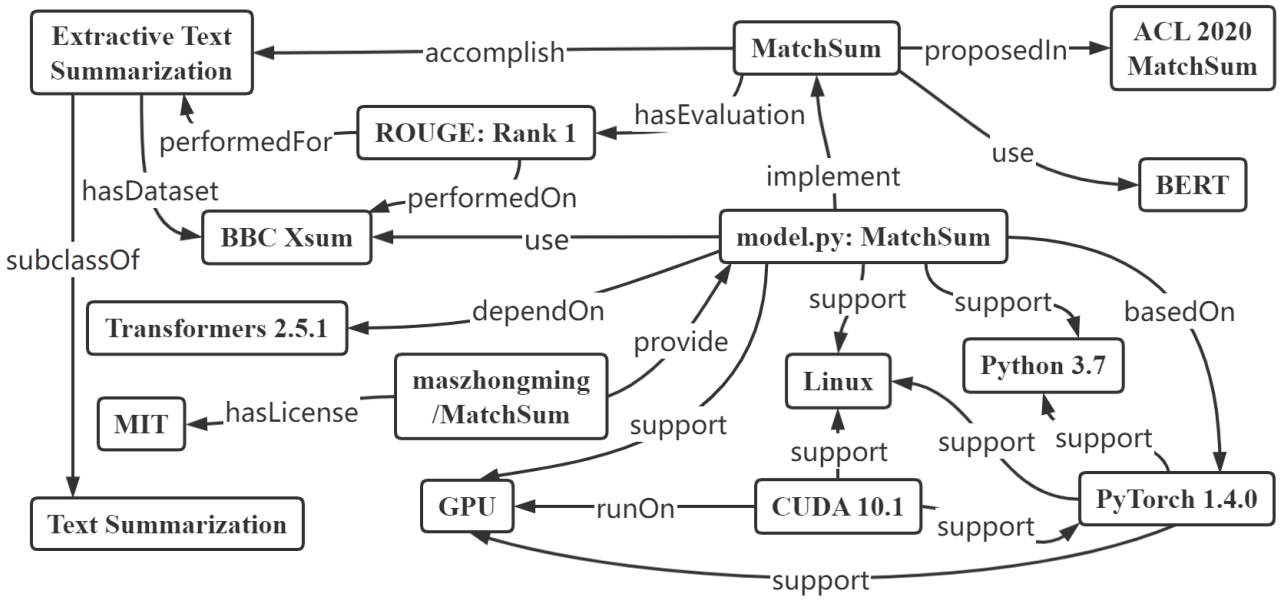


Fig. 5. An Example of AI Task-Model KG

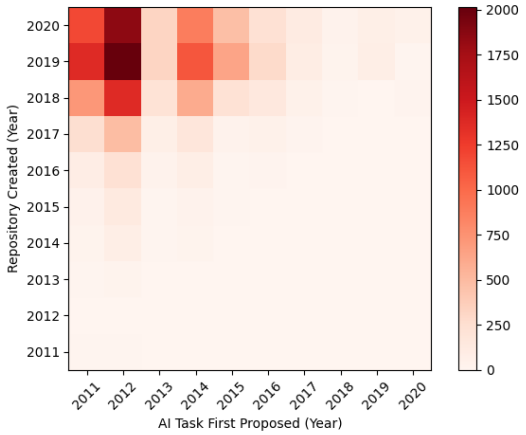


Fig. 6. The Trend of Emerging Implementations of AI Tasks

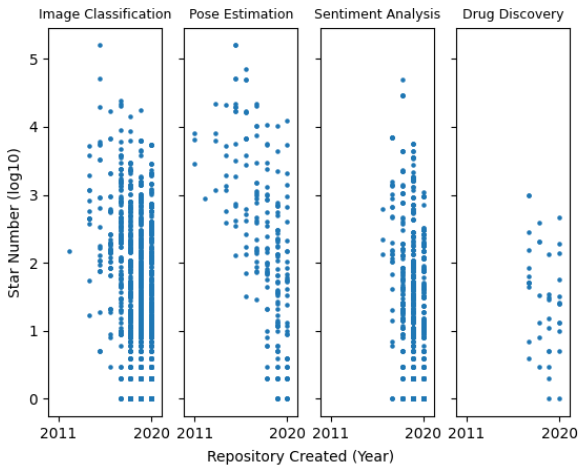


Fig. 7. The Popularities of Repositories of Different Years for Specific AI Tasks

is a fast-developing area. The innovation of AI is reflected in both new AI tasks and new models and implementations for existing AI tasks. Based on our AI task-model KG we can analyze the trend of emerging implementations of AI tasks. For an AI task, new ML/DL models or implementations may constantly emerge. The corresponding repositories have different popularities. For this purpose, we can analyze the popularities of repositories of different years for specific AI tasks.

Results. Figure 6 shows a heat map reflecting the trend of emerging implementations of AI tasks. The analysis considers the 618 second-level tasks (*i.e.*, the direct children of the first-level tasks such as CV and NLP) in the AI task hierarchy of the KG. Each block in the heat map indicates the number of new implementations for AI tasks proposed in different years. For each AI task, we can obtain the time when it first appears in our dataset from the publication time of the first paper for it. For each implementation of an AI task, we obtain the time of its emergence from the creation time of the corresponding open-source repository. It can be seen that a large part of the ML/DL model implementations are targeted at some AI tasks that first appear during 2011-2014 (especially 2011-2012). This trend is consistent with the explosion of deep learning during 2011-2012. A large number of papers using deep learning to accomplish specific AI tasks were published during that time. Many of the targeted AI tasks are classic tasks that attract constant attention. The number of emerging ML/DL model implementations keeps growing and starts an explosive growth from 2018. It is interesting to notice that most of the emerging implementations during 2018-2020 are targeted at those classic tasks that first appear during 2011-2014 (especially 2011-2012). The results suggest, to some extent, that classic tasks attract constant attention with many new implementations, while emerging tasks have much fewer implementations.

Figure 7 shows the popularity of repositories of different years for four specific AI tasks respectively, *i.e.*, image

classification, pose estimation, sentiment analysis, and drug discovery. Each point in the figure represents a repository for the corresponding task and the horizontal axis and the vertical axis represent the creation year and star number (in log10 scale) of the repository respectively. The analysis of these four AI tasks presents different patterns. For example, image classification (the first task in Figure 7) is a classic AI task which is very popular and attracts extensive attention. It has many implementing repositories created in different years and there are high-star repositories of different ages. It can also be seen that, although more and more repositories were created in the recent four years, the most popular repositories were created in 2015. In contrast, drug discovery (the fourth task in Figure 7) is a relatively new and less popular task with much fewer implementing repositories. Moreover, it lacks high-star repositories and there is no significant increase over years.

Summary. Based on the AI task-model KG, MLTaskKG supports a variety of different kinds of trend analysis about AI tasks and their implementations. The analysis can help application developers and related researchers to have an overview of the implementing repositories for different AI tasks and the emergence of new ML/DL models and implementations for specific AI tasks.

6 ML/DL Library Recommendation

MLTaskKG supports interactive ML/DL library recommendation as shown in Figure 8.

For an AI task given by the user, MLTaskKG first retrieves the tasks in the KG, finds the most relevant task, and recommends the libraries that implement the task. We calculate the similarity between the task given by the user and the tasks collected in the KG based on the Word2Vec model (see Section 4.2.3). We then choose the most similar task in the KG as the matched task, identify the ML/DL models accomplishing the task, and return all the libraries implementing the models. To help users to enter a proper query for the desired AI task, MLTaskKG implements several query suggestion strategies, including task completion, parent task and subtask recommendation, and relevant task recommendation. For example, "text classification" will be suggested for "text"; "multi-label text classification" will be suggested for "text classification"; "document classification" will be suggested for "documentation classification".

The recommended libraries are ranked and listed with various information from different sources (*e.g.,* papers, models, and repositories), *e.g.,* repository name and link, star/fork number, model name, citation, and performance ranking. Based on the implementation knowledge of the recommended libraries, MLTaskKG further supports the user to filter the libraries from different dimensions, *i.e.,* hardware, operating system, programming language, ML/DL framework, ML/DL component, and dataset. The options for filtering are generated based on the current set of recommended libraries. For example, only the programming languages supported by the current recommended libraries are provided as options of programming language.

7 Evaluation

MLTaskKG links AI tasks, ML/DL models, and open-source libraries by constructing a knowledge graph. The resulting AI task-model KG can help to analyze the trends of AI tasks and their implementations and recommended libraries for specific AI tasks. To evaluate the effectiveness of MLTaskKG, we conduct a series of experimental studies to answer the following research questions.

RQ1: What is the intrinsic quality of the resulting AI task-model KG?

RQ2: How effective is MLTaskKG in helping developers find suitable libraries for specific AI tasks?

7.1 RQ1: Intrinsic Quality

We evaluate the intrinsic quality of the AI task-model KG by assessing the correctness of the tuples in the KG. A tuple can be a relationship between two entities (*e.g., <MatchSum, accomplish, text summarization>*) or an attribute value of an entity (*e.g., <MatchSum, hasTrainedModel, true>*).

7.1.1 Design

Similar to previous studies [35], [36], [37], we adopt a sampling method [38] to ensure that ratios observed in the sample generalize to the population within a certain confidence interval at a certain confidence level. For a confidence interval of 10 at a 95% confidence level, the required sample size is 96. Thus, for each type of relationship and attribute, we randomly sample 96 tuples from the KG. As a result, we randomly sample 1,632 tuples for 17 types of relationships and attributes. Note that we do not sample tuples that are extracted from PapersWithCode data dump files or GitHub repository metadata as they are intrinsically correct.

We invite four MS students (not affiliated with this work) familiar with ML/DL to annotate the sampled tuples independently. The four students have taken at least two courses on AI and have at least one year of AI development experience. For each tuple they annotate it to be correct or not based on the corresponding knowledge sources (*e.g.,* ReadMe files). If their annotations are different, a third student is assigned to give an additional annotation to resolve the conflict by a majority-win strategy.

7.1.2 Results

The Cohen's Kappa agreement of the first two annotators is 0.742 and the squared Kappa is 0.551, indicating moderate agreement. Disagreements usually occur in two situations, 1) when students lack the corresponding background knowledge about the entities involved in the tuples (*e.g.,* AI tasks, ML/DL models), or 2) when the tuples are extracted from very lengthy knowledge sources (*e.g.,* papers, code, ReadMe files) and students ignore key information.

The results for each type of relationship and attribute are shown in Table 3. According to the annotations, 1,514 tuples are correct and 118 are not, making a correctness of 92.8%. Among all relationship types and attributes, the accuracy of the relationships "implement", "accomplish" and "subclassOf" is relatively low. We analyze the data and find that most errors are rooted in task-model knowledge extraction from ML/DL papers (see Section 4.2.2), *i.e.,* the

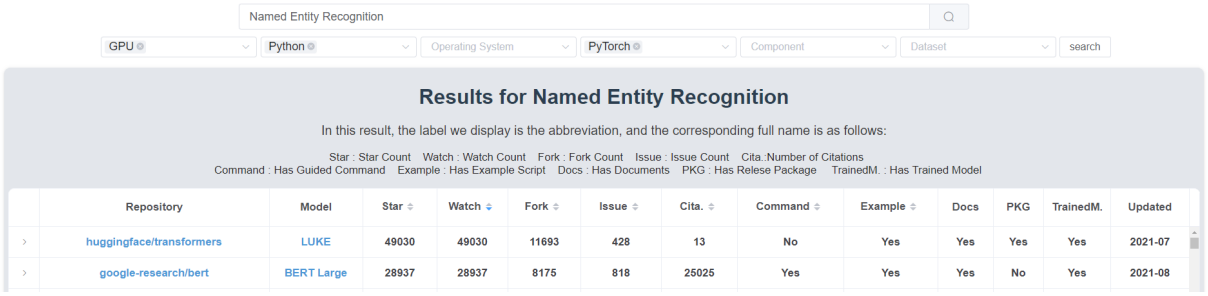


Fig. 8. User Interface of ML/DL Library Recommendation

TABLE 3
Accuracy of Relationships and Attributes

Type	Accuracy	Type	Accuracy	Type	Accuracy
subclassOf	81.3%	proposedIn	90.6%	accomplish	82.3%
implement	78.1%	provide	93.8%	use	92.7%
hasLicense	100%	basedOn	100%	dependOn	87.5%
hasDataset	100%	support	100%	hasTrainedModel	94.8%
hasReleasePackage	96.0%	hasCommand	91.7%	hasTest	94.8%
hasDocument	100%	hasExampleScript	90.6%		

BERT-BiLSTM model identifies tasks and models incorrectly from some papers. In future work, we will try to further improve this by training more advanced models with more training data. Moreover, we can train a tuple classification model to identify and filter low-quality tuples from the KG.

7.1.3 Summary

The AI task-model KG is of high quality which is indicated by the correctness of 92.8% for the sampled tuples.

7.2 RQ2: Recommendation Effectiveness

To evaluate the effectiveness of MLTaskKG, we conducted a human study in which participants were asked to use MLTaskKGto find suitable ML/DL libraries for given AI tasks. To the best of our knowledge, there are currently no tools specifically designed for task-oriented ML/DL library recommendation. However, PapersWithCodecan help in this regard as it organizes state-of-the-art models with implementations by task. Therefore, we chose PapersWithCodeas our baseline for comparison. We did not choose to use technical forums such as Stack Overflow or Medium.io as a baseline because they contain a lot of noise, such as questions unrelated to ML/DL, and have limited coverage for AI tasks or ML/DL models. Similarly, we did not use GitHub as a baseline because it lacks structured descriptions for AI tasks or ML/DL models and includes a significant amount of noise, such as ML/DL tutorials.

7.2.1 Design

We randomly select eight questions aimed at seeking for ML/DL libraries from our empirical study data (see Section 3.1) and adapt them into eight ML/DL library retrieval tasks. The adaptation is focused on improving the expression, *e.g.*, adding explanations for abbreviations and technical terms and changing the order of sentences. One of the tasks is described as follows and the complete list of tasks is available in our replication package [27].

I want to develop a deep learning application that can read and understand news, such as reading comprehension on CNN (Cable News Network), any good libraries you can recommend? My framework version is Pytorch 1.7.

We invite 10 MS students to participate in the study. They have some basic knowledge about ML/DL and experience in AI application development, but are not AI researchers or experts. Therefore, they can well represent the target users of MLTaskKG, *i.e.*, AI application developers. We conduct a pre-study survey on their programming experience and mastery of ML/DL knowledge and divide them into two "equivalent" participant groups (PA and PB) based on the survey. On the other hand, we randomly divide the eight tasks into two task groups (TA and TB) as well. The participants in PA complete TA with PapersWithCode and TB with MLTaskKG. The participants in PB complete TB with PapersWithCode and TA with MLTaskKG. For each participant, the tasks are interleaved, one completed with MLTaskKG and one with PapersWithCode.

We provide a pre-study tutorial to help the participants to understand the requirements of the tasks. The tutorial provides training on tools (MLTaskKG and PapersWithCode) and the protocol to make the participants familiar with the tools and process. For each task the participants need to find and submit an ML/DL library (*i.e.*, GitHub repository) that meets the requirements of the task as the result. If they find multiple libraries they need to choose the most suitable one as the result. We record the libraries they submit and their completion time for each task. Note that they can submit nothing if they don't find a suitable library in 10 minutes.

We aggregate all submitted results for the same task and eliminate any duplicates. We then invite another four MS students who are experts in AI application development to assess the satisfaction of each submitted result by answering the question "Does the library satisfy the requirements of the task and is of high quality?" on a 4-points Likert scale (1-disagree; 2-somewhat disagree; 3-somewhat agree; 4-agree).

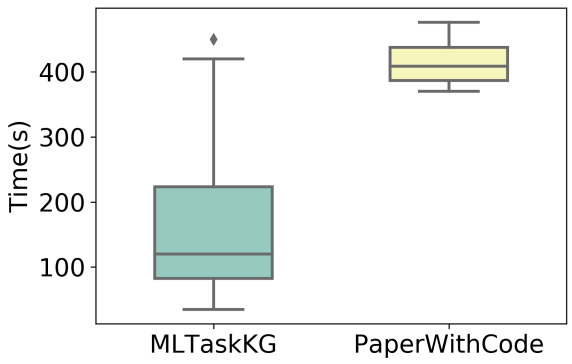


Fig. 9. Comparison of Completion Time between MLTaskKG and PapersWithCode

We ask the four students to rate a submitted result as 4 only if it meets the requirements well and is high-quality; 3 if it satisfies the requirements but is not high-quality; 2 if it doesn't meet the requirements and is high-quality; 1 if it doesn't meet the requirements and is not high-quality. The high satisfaction rating (3 or 4) for a library indicates that it is a suitable choice for completing the task. Each of the four students has taken at least two courses in AI and has at least two years of AI development experience. To eliminate any potential bias, we shuffle the results prior to assessment to ensure that the annotators are unaware of which approach was used to obtain each result.

7.2.2 Results

We obtain 76 results submitted by the participants and the comparison of satisfaction assessment is presented in Table 4. The four failures are all made when PapersWithCode is used. For each failure the satisfaction assessment is recorded as 1. As a result, MLTaskKG received: agree (88), somewhat agree (25), somewhat disagree (20), disagree (19), while PapersWithCode received agree (7), somewhat agree (23), somewhat disagree (73), disagree (41).

We further analyze the completion time of the results that receive only positive feedback (agree or somewhat agree) on satisfaction. Figure 9 shows the comparison between the participants using MLTaskKG and those using PapersWithCode. The participants using MLTaskKG can find more ML/DL libraries with high satisfaction (23 vs. 4) in shorter time (173 seconds vs. 416 seconds on average). We use Welch's t-test [39] to verify the statistical significance of the difference in time and satisfaction between MLTaskKG and PapersWithCode. The results show that both the difference in time and satisfaction are statistically significant (with $p << 0.05$).

We conduct an interview with the participants to learn their feedback on MLTaskKG and PapersWithCode. Interview questions include: 1) Do you think the tool (MLTaskKG or PapersWithCode) is more useful/useless for AI application development and why? 2) What do you like or/dislike about the tool (MLTaskKG or PapersWithCode) and why? 3) Do you have any suggestions for further improvement of the tool? Most of them agree that MLTaskKG provides much better support for AI application development. Some of them mention that PapersWithCode is targeted at AI researchers with the objective of finding papers and SOTA models for comparison. When they seek ML/DL libraries as application developers, they focus more on the matching of tasks and the availability of high-quality implementations for SOTA models. For papers with code PapersWithCode only provides the links to the corresponding repositories. The participants often need to switch back and forth between PapersWithCode and multiple repositories for comparison. In contrast, MLTaskKG recommends matched libraries together with important information like ML/DL framework and programming language. Moreover, the filtering mechanism provided by MLTaskKG can help them quickly filter out unsuitable recommendations. The participants also give us some suggestions for further improvement, for example providing sample code of implementing an AI task with the recommended library.

7.2.3 Summary

Using MLTaskKG, the participants can find ML/DL libraries with 68.4% higher satisfaction and using 47.6% shorter time compared with those using PapersWithCode.

8 Threats to Validity

Both the empirical study and the evaluation involve data annotation, thus a common threat to the internal validity of them is the subjective judgment of the annotators. To alleviate the threat, we follow commonly used data analysis principles, for example multiple annotators, conflict resolution, and reporting agreement coefficients, where appropriate. Since we make our data available for replication, the data can be further evolved and corrected (if needed) by other researchers. Another threat is that our data annotation has been done by students, not professional AI application developers. To alleviate this threat we reported their AI learning and development experience. The invited students should be qualified for our experiments.

Another potential threat to the validity of our KG is related to its completeness. Although we relied on PapersWithCode and AwesomeLists as the initial sources for collecting tasks and models, we acknowledge that they only cover a small subset of the available tasks and models. To address this limitation, we further extracted more tasks and models by analyzing a large number of AI papers that were automatically crawled. While we have made efforts to collect and curate as many papers as possible, our current implementation of MLTaskKG only collects papers from a limited number of AI conferences and journals, which may cause us to miss tasks, models, and libraries from certain AI fields. In addition, the evaluation of the KG only focuses on the correctness of the existing relationships, but we cannot guarantee that we have captured all possible relationships or entities related to a task. Therefore, there may be gaps in the knowledge graph that we have not yet discovered. We acknowledge these limitations and believe that future work could focus on improving the completeness of the KG by expanding the sources of papers and incorporating more expert knowledge to supplement our current approach. Furthermore, while we have made efforts to optimize the performance of each component during the implementation,

TABLE 4
Comparison of Satisfaction Assessment between MLTaskKG and PapersWithCode

Approach	#Agree	#Somewhat Agree	#Somewhat Disagree	#Disagree
MLTaskKG	88	25	20	19
PapersWithCode	7	23	73	41

the absence of a thorough evaluation of each component separately may pose a potential threat to the overall quality of the KG. However, we believe that the overall performance of the KG is a good indicator of the effectiveness of the intermediate components. Future work could focus on evaluating the performance of each individual component to mitigate this potential threat and further improve the accuracy and efficiency of the KG construction process.

A common threat to the external validity is the limited number of subjects (*e.g.,* questions, tasks, and participants) considered in the empirical study and the evaluation. We only analyze 283 ML/DL library seeking questions in the empirical study and conduct eight library seeking tasks with 10 participants in the evaluation of recommendation effectiveness. Thus the findings of the empirical study and evaluation may not be generalized to broader AI application development scenarios in practice. Our current implementation of MLTaskKG only collects the papers of a part of the AI conferences, thus may miss tasks, models, and libraries of some AI fields. Extending our implementation and analysis to more AI fields and tasks may lead to the identification of additional factors, but will not invalidate the results we have obtained.

9 Related Work

With the rapid development of AI technologies and applications, more and more researchers focus on software engineering issues of AI application development [40], including software architecture [41], implementation [42], [43], [44], debugging/testing [45], [46], [47], [48], [49], deployment [43], [50], [51], artifact traceability [52], and technical debt [53]. Some researches are focused on the usage of ML/DL cloud APIs [54], [55] and backward compatibility issues [56]. Zhang *et al.* [42] conduct an empirical study on DL-related questions on Stack Overflow and find that program crashes, model deployment, and implementation are the top three topics of the questions. Different from their work, our work is focused on a specific type of questions, *i.e.,* ML/DL library seeking, and the corresponding support of recommendation.

Knowledge graphs as a representation of structured human knowledge have drawn great attention from both the academia and the industry, which have been applied in many fields such as medicine and finance [57], [58]. Some researchers in the field of software engineering have constructed different types of knowledge graphs for different purposes, such as knowledge graphs for bugs [59], domain terminology [36], API caveats [35], API concept and descriptive knowledge [60], API comparison [37], and software development tasks [61]. Compared with these knowledge graphs, AI task-model KG is a new type of knowledge graphs focusing on the relationships between AI tasks, ML/DL models, and implementation libraries.

In recent years, there have been approaches proposed for API library recommendation by mining app-library usage patterns [62], [63], [64], [65], [66], applying collaborative filtering [67], [68], [69], and training graph neural network [70]. Different from these approaches, MLTaskKG focuses on a specific type of libraries (*i.e.,* ML/DL libraries) and makes a recommendation based on the links between AI tasks, ML/DL models, and the libraries. Shao *et al.* [71] propose paper2repo, an approach that recommends relevant GitHub repositories for a given paper. The approach matches repositories with papers based on joint embeddings of papers and repositories. It does not consider the relationships between AI tasks, ML/DL models, and repositories and cannot support the recommendation of repositories for specific AI tasks. Cao *et al.* [72] present a knowledge graph based approach that can recommend ML models for a given ML dataset. The KG constructed in [72] is a small subset of ours and lacks knowledge specifically captured for developers. Their KG is constructed by parsing PapersWithCode data and includes only knowledge about papers, models, and datasets. Our KG includes knowledge extracted from multiple sources (*e.g.,* PapersWithCode, GitHub repositories, Awesome Lists) and application development knowledge about ML/DL libraries (*e.g.,* environmental dependencies and supporting resources). Their approach retrieves relevant ML/DL models for a given dataset in their KG, while MLTaskKG supports task-oriented ML/DL library recommendation with multi-dimensional comparison.

With the increasing number of scientific publications, including those in the AI domain, many researchers have turned their attention to constructing academic KGs [73], [74], [75], [76]. For example, Färber [74] proposed the Microsoft Academic Knowledge Graph (MAKG), which contains over eight billion triples of information about scientific publications and related entities, such as authors, institutions, journals, and fields of study. However, our KG differs from academic KGs in that it focuses specifically on the AI field and includes information not only about papers but also about ML/DL models and implementing libraries. While some researchers have used academic KGs to design different approaches for recommending papers [77], [78], [79], our work specifically focuses on recommending ML/DL libraries rather than papers.

10 Conclusions and Future Work

Although many ML/DL models and implementations are available, it is not easy for application developers to find ML/DL libraries that are suitable for their AI tasks. Aiming at the problem we conduct an empirical study to understand the requirements of application developers for ML/DL libraries and propose a task-oriented ML/DL library recommendation approach, called MLTaskKG. MLTaskKG constructs a knowledge graph that captures AI tasks, ML/DL models,

model implementations, and their relationships. Based on the knowledge graph, MLTaskKG recommends ML/DL libraries for developers by matching their requirements on tasks, model characteristics, and implementation information. Our evaluation confirms the quality of the resulting knowledge graph and the effectiveness of MLTaskKG for task-oriented trend analysis and recommendation of ML/DL libraries. Our future work will be focused on integrating more knowledge and resources about AI tasks and ML/DL libraries, including background knowledge of AI tasks, and ML/DL library tutorials, sample code, and problem discussions.

11 Data Availability

All the data and results of the work can be found in our replication package [27].

Acknowledgment

This work is supported by National Natural Science Foundation of China under Grant No. 61972098.

References

[1] Z. Duric, W. D. Gray, R. Heishman, F. Li, A. Rosenfeld, M. J. Schoelles, C. Schunn, and H. Wechsler, "Integrating perceptual and cognitive modeling for adaptive and intelligent human-computer interaction," *Proc. IEEE*, vol. 90, no. 7, pp. 1272–1289, 2002.

[2] S. A. Abdul-Kader and J. Woods, "Survey on chatbot design techniques in speech conversation systems," *International Journal of Advanced Computer Science and Applications*, vol. 6, no. 7, 2015.

[3] J. L. Z. Montenegro, C. A. da Costa, and R. da Rosa Righi, "Survey of conversational agents in health," *Expert Systems with Applications*, vol. 129, pp. 56–67, 2019.

[4] Y. Song, S. Dixon, and M. Pearce, "A survey of music recommendation systems and future perspectives," in *9th International Symposium on Computer Music Modeling and Retrieval*, vol. 4. Citeseer, 2012, pp. 395–410.

[5] S. Zhang, L. Yao, A. Sun, and Y. Tay, "Deep learning based recommender system: A survey and new perspectives," *ACM Computing Surveys (CSUR)*, vol. 52, no. 1, pp. 1–38, 2019.

[6] L. Yue, D. Tian, W. Chen, X. Han, and M. Yin, "Deep learning for heterogeneous medical data analysis," *World Wide Web*, vol. 23, no. 5, pp. 2715–2737, 2020.

[7] M. Yang, S. Nazir, Q. Xu, and S. Ali, "Deep learning algorithms and multicriteria decision-making used in big data: A systematic literature review," *Complex.*, vol. 2020, pp. 2 836 064:1–2 836 064:18, 2020.

[8] C. C. Aggarwal and C. Zhai, "A survey of text classification algorithms," in *Mining Text Data*. Springer, 2012, pp. 163–222.

[9] K. Kowsari, K. J. Meimandi, M. Heidarysafa, S. Mendu, L. E. Barnes, and D. E. Brown, "Text classification algorithms: A survey," *Inf.*, vol. 10, no. 4, p. 150, 2019.

[10] R. Jafri and H. R. Arabnia, "A survey of face recognition techniques," *J. Inf. Process. Syst.*, vol. 5, no. 2, pp. 41–68, 2009.

[11] M. Chihaoui, A. Elkefi, W. Bellil, and C. B. Amar, "A survey of 2d face recognition techniques," *Comput.*, vol. 5, no. 4, p. 21, 2016.

[12] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International journal of Remote sensing*, vol. 28, no. 5, pp. 823–870, 2007.

[13] S. S. Nath, G. Mishra, J. Kar, S. Chakraborty, and N. Dey, "A survey of image classification methods and techniques," in *2014 International conference on control, instrumentation, communication and computational technologies (ICCICCT)*. IEEE, 2014, pp. 554–557.

[14] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika, J. C. Niebles, M. Sellitto *et al.*, "The ai index 2021 annual report," *arXiv preprint arXiv:2103.06312*, 2021.

[15] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez, "A survey on deep learning techniques for image and video semantic segmentation," *Applied Soft Computing*, vol. 70, pp. 41–65, 2018.

[16] V. Wiley and T. Lucas, "Computer vision and image processing: a paper review," *International Journal of Artificial Intelligence Research*, vol. 2, no. 1, pp. 29–36, 2018.

[17] Y. Goldberg, *Neural Network Methods for Natural Language Processing*, ser. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2017.

[18] D. W. Otter, J. R. Medina, and J. K. Kalita, "A survey of the usages of deep learning for natural language processing," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 32, no. 2, pp. 604–624, 2021.

[19] L. Besacier, E. Barnard, A. Karpov, and T. Schultz, "Automatic speech recognition for under-resourced languages: A survey," *Speech Commun.*, vol. 56, pp. 85–100, 2014.

[20] M. Malik, M. K. Malik, K. Mehmood, and I. Makhdoom, "Automatic speech recognition: a survey," *Multim. Tools Appl.*, vol. 80, no. 6, pp. 9411–9457, 2021.

[21] S. M. Grigorescu, B. Trasnea, T. T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *J. Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[22] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58 443–58 469, 2020.

[23] X. Zhang and M. Lapata, "Sentence simplification with deep reinforcement learning," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, September 9-11, 2017 Copenhagen, Denmark*. Association for Computational Linguistics, 2017, pp. 584–594.

[24] L. Martin, É. de la Clergerie, B. Sagot, and A. Bordes, "Controllable sentence simplification," in *Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, May 11-16, 2020*.

[25] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, April 3-7, 2017, Valencia, Spain, Volume 2: Short Papers*, M. Lapata, P. Blunsom, and A. Koller, Eds. Association for Computational Linguistics, 2017, pp. 427–431.

[26] StackOverflow, "Stack overflow data dump version from march 1, 2021," https://archive.org/download/stackexchange/.

[27] (2021) Replication package. [Online]. Available: https://mltaskkg.github.io/

[28] M. L. McHugh, "Interrater reliability: the kappa statistic," *Biochemia medica: Biochemia medica*, vol. 22, no. 3, pp. 276–282, 2012.

[29] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, June 2-7, 2019, Minneapolis, MN, USA, Volume 1 (Long and Short Papers)*, 2019, pp. 4171–4186.

[30] I. Beltagy, K. Lo, and A. Cohan, "Scibert: Pretrained language model for scientific text," in *EMNLP*, 2019.

[31] H. Nakayama, T. Kubo, J. Kamura, Y. Taniguchi, and X. Liang, "doccano: Text annotation tool for human," 2018, software available from https://github.com/doccano/doccano. [Online]. Available: https://github.com/doccano/doccano

[32] (2021) gensim. [Online]. Available: https://radimrehurek.com/gensim/

[33] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of jaccard coefficient for keywords similarity," in *Proceedings of the international multiconference of engineers and computer scientists*, vol. 1, no. 6, 2013, pp. 380–384.

[34] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo, "Categorizing the content of github README files," *Empir. Softw. Eng.*, vol. 24, no. 3, pp. 1296–1327, 2019.

[35] H. Li, S. Li, J. Sun, Z. Xing, X. Peng, M. Liu, and X. Zhao, "Improving API caveats accessibility by mining API caveats knowledge graph," in *2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, September 23-29, 2018, Madrid, Spain*, 2018, pp. 183–193.

[36] C. Wang, X. Peng, M. Liu, Z. Xing, X. Bai, B. Xie, and T. Wang, "A learning-based approach for automatic construction of domain glossary from source code and documentation," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, 2019, pp. 97–108.

[37] Y. Liu, M. Liu, X. Peng, C. Treude, Z. Xing, and X. Zhang, "Generating concept based API element comparison using a knowledge graph," in *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, September 21-25, 2020, Melbourne, Australia*. IEEE, 2020, pp. 834–845.

[38] R. Singh and N. S. Mangat, *Elements of survey sampling*. Springer Science & Business Media, 2013, vol. 15.

[39] B. L. Welch, "The generalization of student's problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, pp. 28–35, 1947.

[40] G. Lorenzoni, P. S. C. Alencar, N. M. do Nascimento, and D. D. Cowan, "Machine learning model development from a software engineering perspective: A systematic literature review," *CoRR*, vol. abs/2102.07574, 2021. [Online]. Available: https://arxiv.org/abs/2102.07574

[41] A. Serban and J. Visser, "An empirical study of software architecture for machine learning," *CoRR*, vol. abs/2105.12422, 2021. [Online]. Available: https://arxiv.org/abs/2105.12422

[42] T. Zhang, C. Gao, L. Ma, M. R. Lyu, and M. Kim, "An empirical study of common challenges in developing deep learning applications," in *30th IEEE International Symposium on Software Reliability Engineering, ISSRE 2019, October 28-31, 2019, Berlin, Germany*. IEEE, 2019, pp. 104–115.

[43] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, "An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, November 11-15, 2019, San Diego, CA, USA*. IEEE, 2019, pp. 810–822.

[44] M. Alshangiti, H. Sapkota, P. K. Murukannaiah, X. Liu, and Q. Yu, "Why is developing machine learning applications challenging? A study on stack overflow posts," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2019, September 19-20, 2019, Porto de Galinhas, Recife, Brazil*. IEEE, 2019, pp. 1–11.

[45] F. Thung, S. Wang, D. Lo, and L. Jiang, "An empirical study of bugs in machine learning systems," in *23rd IEEE International Symposium on Software Reliability Engineering, ISSRE 2012, November 27-30, 2012, Dallas, TX, USA*. IEEE Computer Society, 2012, pp. 271–280.

[46] M. J. Islam, G. Nguyen, R. Pan, and H. Rajan, "A comprehensive study on deep learning bug characteristics," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, August 26-30, 2019, Tallinn, Estonia*. ACM, 2019, pp. 510–520.

[47] Y. Zhang, Y. Chen, S. Cheung, Y. Xiong, and L. Zhang, "An empirical study on tensorflow program bugs," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2018, July 16-21, 2018, Amsterdam, The Netherlands*. ACM, 2018, pp. 129–140.

[48] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *42nd International Conference on Software Engineering, Seoul,ICSE 2020, 27 June - 19 July, 2020, South Korea*. ACM, 2020, pp. 1110–1121.

[49] M. J. Islam, R. Pan, G. Nguyen, and H. Rajan, "Repairing deep neural networks: fix patterns and challenges," in *42nd International Conference on Software Engineering, Seoul, ICSE 2020, June 27 - July 19, 2020, South Korea*. ACM, 2020, pp. 1135–1146.

[50] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, "A comprehensive study on challenges in deploying deep learning based software," in *28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2020, November 8-13, 2020, Virtual Event, USA*. ACM, 2020, pp. 750–762.

[51] Y. Ma, D. Xiang, S. Zheng, D. Tian, and X. Liu, "Moving deep learning into web browser: How far can we go?" in *The World Wide Web Conference, WWW 2019, May 13-17, 2019, San Francisco, CA, USA*. ACM, pp. 1234–1244.

[52] A. T. Njomou, A. J. B. Africa, B. Adams, and M. Fokaefs, "MSR4ML: reconstructing artifact traceability in machine learning repositories," in *28th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2021, Honolulu, HI, USA, March 9-12, 2021*. IEEE, 2021, pp. 536–540.

[53] Y. Tang, R. Khatchadourian, M. Bagherzadeh, R. Singh, A. Stewart, and A. Raja, "An empirical study of refactorings and technical debt in machine learning systems," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 238–250.

[54] A. Cummaudo, R. Vasa, S. Barnett, J. C. Grundy, and M. Abdelrazek, "Interpreting cloud computer vision pain-points: a mining study of stack overflow," in *42nd International Conference on Software Engineering, Seoul,ICSE 2020, South Korea, 27 June - 19 July, 2020*. ACM, 2020, pp. 1584–1596.

[55] C. Wan, S. Liu, H. Hoffmann, M. Maire, and S. Lu, "Are machine learning cloud APIs used correctly?" in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, 22-30 May 2021, Madrid, Spain*. IEEE, 2021, pp. 125–137.

[56] M. Srivastava, B. Nushi, E. Kamar, S. Shah, and E. Horvitz, "An empirical analysis of backward compatibility in machine learning systems," in *The 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD 2020, August 23-27, 2020, Virtual Event, CA, USA*. ACM, 2020, pp. 3272–3280.

[57] X. Zou, "A survey on application of knowledge graph," in *Journal of Physics: Conference Series*, vol. 1487, no. 1. IOP Publishing, 2020, p. 012016.

[58] S. Ji, S. Pan, E. Cambria, P. Marttinen, and S. Y. Philip, "A survey on knowledge graphs: Representation, acquisition, and applications," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 2, pp. 494–514, 2021.

[59] L. Wang, X. Sun, J. Wang, Y. Duan, and B. Li, "Construct bug knowledge graph for bug resolution: poster," in *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, May 20-28, 2017 - Companion Volume, Buenos Aires, Argentina*. IEEE Computer Society, 2017, pp. 189–191.

[60] M. Liu, X. Peng, A. Marcus, Z. Xing, W. Xie, S. Xing, and Y. Liu, "Generating query-specific class API summaries," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2019, August 26-30, 2019, Tallinn, Estonia*. ACM, 2019, pp. 120–130.

[61] M. Liu, X. Peng, A. Marcus, C. Treude, J. Xie, H. Xu, and Y. Yang, "How to formulate specific how-to questions in software development?" in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 306–318.

[62] M. Chouchen, A. Ouni, and M. W. Mkaouer, "Androlib: Third-party software library recommendation for android applications," in *Reuse in Emerging Software Engineering Practices - 19th International Conference on Software and Systems Reuse, ICSR 2020,December 2-4, 2020, Hammamet, Tunisia*, ser. Lecture Notes in Computer Science, vol. 12541. Springer, 2020, pp. 208–225.

[63] A. Ouni, R. G. Kula, M. Kessentini, T. Ishio, D. M. Germán, and K. Inoue, "Search-based software library recommendation using multi-objective optimization," *Inf. Softw. Technol.*, vol. 83, pp. 55–75, 2017.

[64] M. A. Saied, A. Ouni, H. A. Sahraoui, R. G. Kula, K. Inoue, and D. Lo, "Improving reusability of software libraries through usage pattern mining," *J. Syst. Softw.*, vol. 145, pp. 164–179, 2018.

[65] M. A. Saied and H. A. Sahraoui, "A cooperative approach for combining client-based and library-based API usage pattern mining," in *24th IEEE International Conference on Program Comprehension, ICPC 2016,May 16-17, 2016, Austin, TX, USA*. IEEE Computer Society, 2016, pp. 1–10.

[66] C. Teyton, J. Falleri, and X. Blanc, "Automatic discovery of function mappings between similar libraries," in *20th Working Conference on Reverse Engineering, WCRE 2013, October 14-17, 2013, Koblenz, Germany*. IEEE Computer Society, 2013, pp. 192–201.

[67] P. T. Nguyen, J. D. Rocco, D. D. Ruscio, and M. D. Penta, "Crossrec: Supporting software developers by recommending third-party libraries," *J. Syst. Softw.*, vol. 161, 2020.

[68] Q. He, B. Li, F. Chen, J. Grundy, X. Xia, and Y. Yang, "Diversified third-party library prediction for mobile app development," *IEEE Transactions on Software Engineering*, 2020.

[69] F. Thung, D. Lo, and J. L. Lawall, "Automated library recommendation," in *20th Working Conference on Reverse Engineering, WCRE 2013, October 14-17, 2013, Koblenz, Germany*. IEEE Computer Society, 2013, pp. 182–191.

[70] B. Li, Q. He, F. Chen, X. Xia, L. Li, J. C. Grundy, and Y. Yang, "Embedding app-library graph for neural third party library recommendation," in *29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2021, August 23-28, 2021, Athens, Greece*. ACM, 2021, pp. 466–477.

[71] M. Färber, "Analyzing the github repositories of research papers," in *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries, JCDL 2020, August 1-5, 2020, Virtual Event, China*. ACM, 2020, pp. 491–492.

[72] X. Cao, Y. Shi, H. Yu, J. Wang, X. Wang, Z. Yan, and Z. Chen, "DEKR: description enhanced knowledge graph for machine learning method recommendation," in *The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval,SIGIR 2021, July 11-15, 2021, Virtual Event, Canada*. ACM, 2021, pp. 203–212.
[73] P. Manghi, A. Mannocci, F. Osborne, D. Sacharidis, A. Salatino, and T. Vergoulis, "New trends in scientific knowledge graphs and research impact assessment," *Quantitative Science Studies*, vol. 2, no. 4, pp. 1296–1300, 2022.
[74] M. Färber, "The microsoft academic knowledge graph: A linked data source with 8 billion triples of scholarly data," in *International semantic web conference*. Springer, 2019, pp. 113–129.
[75] L. Pollacci, "Emakg: An enhanced version of the microsoft academic knowledge graph," *arXiv preprint arXiv:2203.09159*, 2022.
[76] J. Liu, J. Ren, W. Zheng, L. Chi, I. Lee, and F. Xia, "Web of scholars: A scholar knowledge graph," in *43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 2153–2156.
[77] B. Wang, Z. Weng, and Y. Wang, "A novel paper recommendation method empowered by knowledge graph: for research beginners," *CoRR*, vol. abs/2103.08819, 2021.
[78] X. Wang, H. Xu, W. Tan, Z. Wang, and X. Xu, "Scholarly paper recommendation via related path analysis in knowledge graph," in *2020 International Conference on Service Science, ICSS 2020, Xining, China, August 24-26, 2020*. IEEE, 2020, pp. 36–43.
[79] J. Chicaiza and P. V. Díaz, "A comprehensive survey of knowledge graph-based recommender systems: Technologies, development, and contributions," *Inf.*, vol. 12, no. 6, p. 232, 2021.

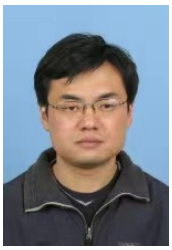
Mingwei Liu received his B.S. degree and Ph.D. degree in Software Engineering from Fudan University in 2017 and 2022, respectively. He currently works as a Postdoctoral Fellow at Fudan University. His research focuses on the intersection of Software Engineering (SE) and Artificial Intelligence (AI), with a particular emphasis on AI4SE and SE4AI. Mingwei leverages advanced AI technologies, including large language models and knowledge graphs, to tackle software engineering challenges and adapt to the evolving landscape of AI systems. He has published over 10 papers in prestigious international journals and conferences, such as TSE, TOSEM, ESEC/FSE, ASE, and ICSME. Mingwei's contributions have been recognized with the IEEE TCSE Distinguished Paper Awards at ICSME 2018. For more information about his work, please visit his website at https://mingwei-liu.github.io/.

Chengyuan Zhao received the bachelor's degree from Chongqing University, in 2019. He is working toward the master's degree in the School of Computer Science, Fudan University, China. His work mainly concerns on intelligent software development.

Xin Peng received the bachelor's and PhD degrees in computer science from Fudan University, in 2001 and 2006, respectively. He is a professor of the School of Computer Science, Fudan University, China. His research interests include data-driven intelligent software development, cloud-native software and AIOps, software engineering for AI and cyber-physical social Systems. His work won the ICSM 2011 Best Paper Award, the ACM SIGSOFT Distinguished Paper Award at ASE 2018/2021 and ICPC 2022, the IEEE TCSE Distinguished Paper Awards at ICSME 2018/2019/2020 and SANER 2023, and the IEEE Transactions on Software Engineering 2018 Best Paper Award.

Haofen Wang is a professor at College of Design & Innovation, Tongji University. Prior to that, He served as CTOs for two well-known AI startups (i.e., Leyan and Gowild). He is also one of the co-founders of OpenKG, the world-largest Chinese open knowledge graph community. He has taken charge of several national AI projects and published more than 100 related papers on top-tier conferences and journals. He developed the first interactive emotional virtual idol in the world. The intelligent assistant he built has answered questions from more than one billion users when they did online shopping. He has also served as deputy directors or chairs for several NGOs like CCF, CIPS and SCS.

Simin Yu received the bachelor's degree from East China Normal University, in 2020. She is studying for a master's degree in the School of Computer Science, Fudan University. Her work mainly concerns on intelligent software development.



Chaofeng Sha received the PhD degree in computer science from Fudan University in 2009. He is a Associate Professor of the School of Computer Science, Fudan University, China. His research interests include machine learning and data mining, AI for SE, and SE for AI.